گزارش پروژه بینایی کامپیوتر

نویسنده: رضا کریم‌زاده

شماره دانشجویی: 98206234

استاد درس: دکتر محمدزاده

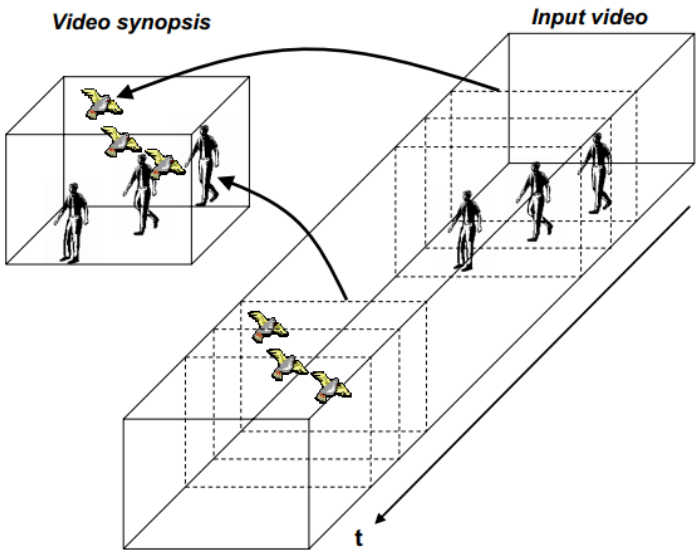
تابستان 99

فهرست

فهرست اشکال

1 مقدمه

در دوربین‌های نظارتی به علت زمان زیادی که تصویر برداری انجام شده عملا بسیار سخت و زمانبر است که کل زمان یک شخص در حال نظارت بر ویدیو باشد بنابراین یک تکنیک لازم است تا بدون از دست دادن جزییات یک ویدیو با زمان بسیار کم از روی ویدیوی اصلی ساخته شود به این کار خلاصه سازی ویدیو[1] گفته می‌شود. به عنوان مثال تصویر زیر را در نظر بگیرید، در ویدیوی اصلی عابرین پیاده و پرندگان در دو زمان متفاوت و دور از یکدیگر حرکت می‌کنند اما در ویدیوی خلاصه شده فریم‌های مربوط به هر دو گروه در چند فریم به صورت خلاصه قرار گرفته است. بنابراین با این روش یک ویدیو با زمان بسیار کوتاه‌تر بدون حذف جزییات خواهیم داشت.



شکل 1-1 ویدیوی اصلی و ویدیوی خلاصه شده

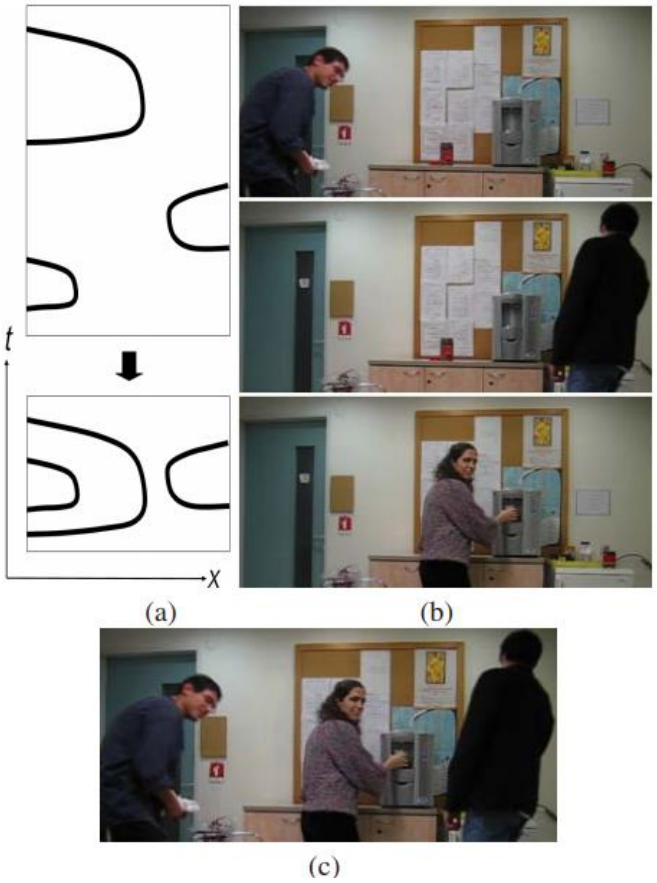
در خلاصه‌سازی ویدیو در کل دو نوع رویکرد وجود دارد:

- در رویکرد اول وقایع کلیدی که می‌تواند نشانگر موضوعیت کلی رخداد باشد ذخیره و در نهایت نشان داده می‌شوند.
- در رویکرد دوم یک مجموعه از ویدیوهای کوتاه شده‌ی مربوط به اشیای متحرک از روی ویدیوی اصلی ساخته می‌شود و در نهایت با ترکیب شدن این ویدیوها یک ویدیوی خلاصه شده بدست می‌آید.

[1] Video Synopsis

رویکرد ما در این پروژه استفاده از گزینه‌ی دوم است. برای این نوع خلاصه سازی یک سازش[۲] بین زمان خلاصه سازی و همپوشانی اشیای در حال حرکت وجود دارد به این صورت که اگر بخواهیم همپوشانی بسیار کم شود باید زمان ویدیوی خلاصه شده افزایش یابد تا اشیای هم پوشان در دو زمان متفاوت وارد شوند.

در شکل زیر یک نمونه از خلاصه‌سازی ویدیو مشاهده می‌شود که سه شی متحرک در زمان‌های مختلف وارد شده‌اند اما همپوشانی مکانی ندارند و در یک زمان نشان داده شده‌اند[1-3].



شکل 2-1 نمونه ای از خلاصه سازی

روند انجام این رویکرد می‌تواند به صورت زیر خلاصه شود:

- استفاده از یکی از روش‌های حذف پس زمینه برای داشتن موقعیت و شکل کلی اشیای متحرک در تصویر

[2] Trade off

- دنبال کردن اشیای متحرک از لحظه‌ی ورود تا لحظه‌ی خروج از کادر تصویر
- استخراج ویدیوی مربوط به هر شی متحرک
- ترکیب ویدیوهای اشیای متحرک برای خلاصه‌سازی نهایی

در ادامه به توضـــیح هر یک از گام‌های بالا و روند اجرای آن و چالش‌هایی که در زمان اجرا وجود دارد پرداخته می‌شود.

2 حذف پس زمینه و استخراج اشیای متحرک

برای حذف پس‌زمینه متدهای مختلفی وجود دارد که در ادامه به تفصیل توضیح داده می‌شود.

2_1 انتخاب فریم اول به عنوان پس زمینه

در بسیاری از خلاصه‌سازی های بلادرنگ[3] برای کم کردن حجم محاسبات و سریع بودن الگوریتم فریم اول را به عنوان پس زمینه استفاده می‌کنند و با کم کردن آن از فریم‌های بعد، اشیای متحرک در ویدیوی اصلی استخراج خواهد شد. در این پروژه به علت آنکه تمام ویدیو موجود است و سیستم بلادرنگ نیست برای افزایش دقت از این متد استفاده نشد.

2_2 استخراج پس زمینه با استفاده از فیلتر میانه و مشخص سازی اشیای متحرک

یکی از متدهای پر استفاده برای استخراج پس زمینه استفاده از میانه‌ی پیکسل‌های ویدیو برای پیکسل منتناظر پس زمینه است. برای این کار از کد زیر استفاده گردید. ابتدا ویدیو خوانده شد و در یک آرایه کلیه‌ی فریم‌های آن قرار داده شد. سپس با استفاده از یک فیلتر میانه‌گیر برای پیکسل‌های پشت سر هم قرار داده شده، پس زمینه استخراج و ذخیره گردید.

```
1.  # extract background and track objects
2.  import cv2
3.  import numpy as np
4.  cap = cv2.VideoCapture('Video1.avi')
5.
```

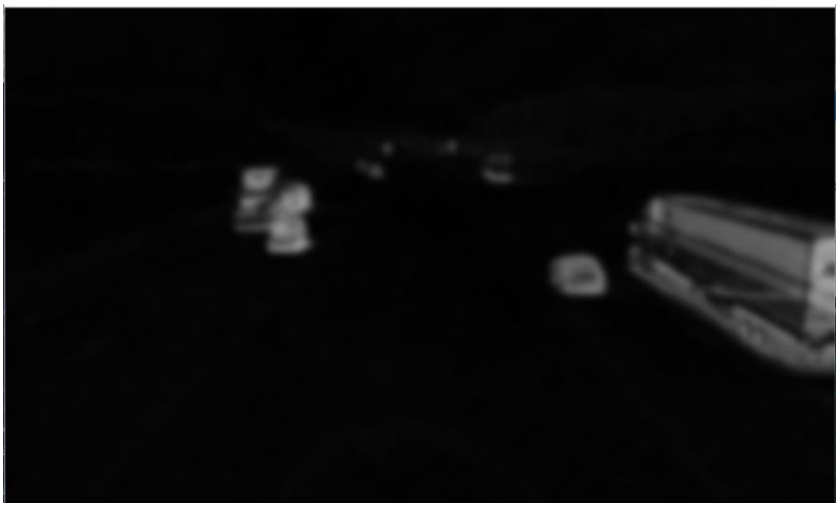
[3] Real Time

```
seq=[]

while True:
    ret_val, frame = cap.read()
    if frame is None:
        break
    seq.append(frame)

# convert to numpy array
seq = np.array(seq)

# calculate median of arrays
background = np.median(seq, axis=0).astype('uint8')
print('background shape: ',background.shape)
cv2.imwrite('background2.jpg',background)
cap.release()
cv2.imshow('background',background)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

نتیجه برای ویدیوی اول به صورت زیر حاصل گردید.

شکل 1-2 پس زمینه ی ویدیوی اول

حال با استفاده از این پس زمینه‌ی استخراج شده، با تفریق آن از فریم‌های دیگر میتوان اشیای متحرک را استخراج نمود. نمونه ای از این کار در شکل زیر مشاهده می‌شود.

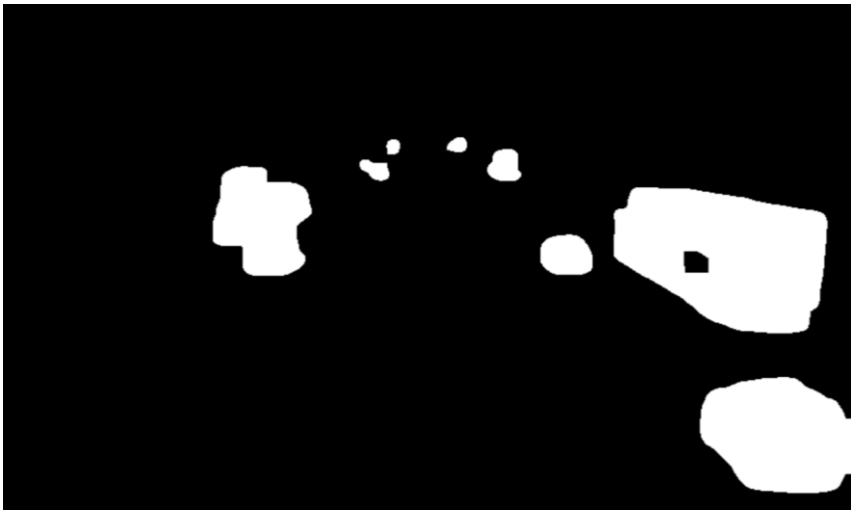


شکل 2-2 اشیای متحرک با حذف پس زمینه

در گام بعد برای دنبال کردن اشیای متحرک باید کانتورهای اطراف این اشیا استخراج شود بنابراین یک سری عملیات برای صفر کردن شدت پس زمینه و یک کردن شدت‌های شی متحرک برای دادن به الگوریتم پیدا کردن کانتور و لازم است. برای این کار از یک ترشلدگذاری باینری استفاده شد. در زیر نتایج خروجی مشاهده می‌شود.

شکل 3-2 تصویر باینری بدست آمده با آستانه گذاری

با توجه به تصویر بالا مشاهده می‌شود که در بعضی لز نواحی سوراخ‌هایی وجود دارد و در برخی موارد دو شی نزدیک به هم متصل گردیده‌اند برای حل این مشکل از عملیات‌های مورفولوژی برای پر کردن سوراخ‌های تصویر باینری و قطع ارتباط اشیای مستقل استفاده گردید.



شکل 4-2 انجام عملیات مورفولوژی بر روی تصویر باینری

مشاهده می‌شود سوراخ‌های موجود در اشیا تا حد خوبی حذف گردیدند اما همچنان شکل آن که دو شی نزدیک به هم چسبیده نشان داده می‌شود وجود دارد. با این اوصاف تصویر بدست آمده را به الگوریتم findContours می‌دهیم و پس از محاسبه‌ی مساحت کانتور و اعمال یک ترشلد برای یافتن کانتورهای بزرگ‌تر از یک حد Bounding Box متناظر هر کانتور استخراج و بر روی شی متحرک رسم گردید.

شکل 5-2 اشیای متحرک مشخص شده با bounding box

همانطور که در بالا مشاهده شد در این روش اشیای نزدیک به هم به عنوان یک شی تلقی می‌شود و مشکل دیگر در صورت وزش باد یا گرد و خاکی که از حرکت ماشین‌ها ایجاد می‌شود یک باکس هم برای آن‌ها در نظر گرفته می‌شود. بنابراین با وجود این مشکلات از تفریق پس زمینه و فریم برای مشخص

کردن اشـیای متحرک خودداری شـد و به دنبال یک متد دیگر برای رفع این مشـکل گشـتیم. در زیر کد مربوط به عملیات بالا مشاهده می‌شود.

```
import cv2
import numpy as np
import imutils

background = cv2.imread('background1.jpg')
cap = cv2.VideoCapture('Video1.avi')
# track objects

while True:
    ret_val, frame = cap.read()
    if frame is None:
        break
    # compute the difference between the current frame and
    # background
    diff =cv2.absdiff(frame,background)
    gray_diff = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
    #blur it
    gray_diff = cv2.GaussianBlur(gray_diff, (21, 21), 0)
    #thresholding for find moving objects
    thresh = cv2.threshold(gray_diff, 20, 255, cv2.THRESH_BINARY)[1]
    # dilate the thresholded image to fill in holes, then find contours
    # on thresholded image
    kernel = np.ones([7,7])
    dilate = cv2.dilate(thresh, kernel, iterations=2)
    erode = cv2.erode(dilate, kernel, iterations=2)
    cnts = cv2.findContours(erode.copy(), cv2.RETR_TREE,
        cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    # loop over the contours
    for c in cnts:
        # if the contour is too small, ignore it
        if cv2.contourArea(c) < 5:
            continue
        # compute the bounding box for the contour, draw it on the frame,
        # and update the text
        (x, y, w, h) = cv2.boundingRect(c)
#        find color of moving object
        bb = frame[y+h//4:y+3*h//4,x+w//4:x+3*w//4,:];
        color = np.mean(bb,axis=0)
        color = np.floor(np.mean(color, axis=0))
        # just select middle point intensity
#        color = np.array([frame[y+h//2,x+w//2,0], frame[y+h//2,x+w//2,1], frame[
y+h//2,x+w//2,2]],dtype='float')
        cv2.rectangle(frame, (x, y), (x + w, y + h), (color[0], color[1], color[2
]), 2)

    cv2.imshow('thresh',frame)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
```

3_2 استفاده از شار نوری برای دنبال کردن اشیا

در این متد از الگوریتم شـار نوری calcOpticalFlowFarneback اسـتفاده گردید که با اسـتفاده از دو فریم متوالی زوایه و اندازه‌ی شـار را مشـخص می‌سـازد در نهایت با ترکیب وزن دار تصـویر رنگی سـاخته شده بر اساس شار نوری و فریم‌های ویدیو اشیای متحرک مشخص گردید.

شکل 6-2 شار نوری ویدیو

مشاهده می‌شود علاوه بر اینکه اشیای متحرک شناسایی می‌شود، نویز هم به علت وزش باد یا گرد و غبار در اثر حرکت خودروها نیز شناسایی می‌شود. در برخی از فریم‌ها مانند شکل زیر نویز زیادی را شناسایی کرده است.

شکل 7-2 شار نوری بسیار نویزی برای یکی از فریم ها

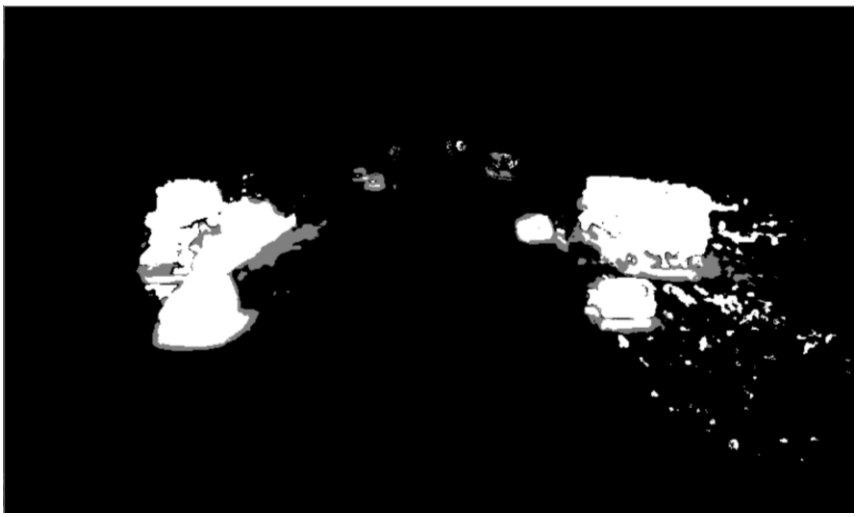
بنابراین با این نتایج نمی‌توان از این روش نتیجه‌ی خوبی گرفت و باید به دنبال یک رویکرد دیگر بود. در ادامه کد مربوط به بدست آوردن شار نوری ملاحظه می‌شود.

```
import cv2
import numpy as np
import imutils
#%%
cap = cv2.VideoCapture('Video1.avi')

seq=[]

while True:
    ret_val, frame = cap.read()
    if frame is None:
        break
    seq.append(frame)

#%% optical flow
prev_gray = cv2.cvtColor(seq[0], cv2.COLOR_BGR2GRAY)
prev_gray = cv2.GaussianBlur(prev_gray, (9, 9), 0)
# Create mask
mask = np.zeros_like(background)
# Set image saturation to maximum value as we do not need it
mask[:,:, 1] = 255
#
for i in range(len(seq)//5-1):
    # compute the difference between the current frame and
    # background
    frame = seq[i+1]

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (9, 9), 0)

    # Calculate dense optical flow by Farneback method
    flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None, pyr_scale = 0.5, l
evels = 5, winsize = 11, iterations = 5, poly_n = 5, poly_sigma = 1.1, flags = 0)

     # Compute the magnitude and angle of the 2D vectors
    magnitude, angle = cv2.cartToPolar(flow[:,:, 0], flow[:,:, 1])
     # Set image hue according to the optical flow direction
    mask[:,:,0] = angle * 180 / np.pi / 2
     # Set image value according to the optical flow magnitude (normalized)
    magnitude = cv2.threshold(magnitude, 10, 255, cv2.THRESH_TRUNC)[1]
    mask[:,:,2] = cv2.normalize(magnitude, None, 0, 255, cv2.NORM_MINMAX)
    #mm = cv2.threshold(mag, 60, 255, cv2.THRESH_BINARY)[1]
     # Convert HSV to RGB (BGR) color representation
    rgb = cv2.cvtColor(mask, cv2.COLOR_HSV2BGR)
    dense_flow = cv2.addWeighted(frame, 1,rgb, 2, 0)
    cv2.imshow('Dense optical flow', dense_flow)

    prev_gray = gray
    k = cv2.waitKey(1) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
```

4_2 استخراج پس زمینه بر اساس مدل‌های گوسی مختلط[4]

یکی از رویکردهای استخراج پس زمینه استفاده از مدل گوسی مختلط است به این صورت که هر تصویر با چند گوسی تخمین زده می‌شود و در نهایت با استخراج گوسی های متناسب با اشیای متحرک پس زمینه حذف می‌گردد.

یکی از ویژگی‌های خوب این روش امکان تشخیص و حذف سایه‌ی اجسام است. خوشبختانه این روش توسط کتابخانه‌ی open cv پیاده سازی شده است و نتایج خوبی ایجاد می‌کند. بنابراین با استفاده از این متد یک تصویر خاکستری از اشیای متحرک بدست می‌آید. نمونه ای از این خروجی در تصویر زیر مشاهده می‌شود.



شکل 8-2 تصویر خروجی پس از حذف پس زمینه

در ادامه با استفاده از عملیات‌های مورفولوژی closing و erosion بر روی تصویر خروجی و یک آستانه‌گذاری مناسب، تصویر باینری نهایی ایجاد گردید.

[4] Gaussian Mixture Model

شکل 9-2 تصویر باینری اشیای متحرک

در ادامه با دادن این تصـــویر بایتری به الگوریتم پیدا کننده‌ی کانتور، کانتورهای متناظر اشـــیا اســـتخراج گردید و در ادامه‌ی پردازش چون در برخی از این کانتورها ناحیه‌ی تعقر موجود بود با اسـتفاده از دسـتور convexHull این نواحی محدب گردید و در نهایت Bounding Box مربوط به هر ناحیه اســـتخراج گردید نتیجه به صورت زیر مشاهده می‌گردد.

شکل 10-2 باکس های استخراج شده

مشاهده می‌شود این الگوریتم اشیای نزدیک به هم را به خوبی استخراج کرده و حساسیت کمتری نسبت به نویز دارد. بنابراین در ادامه‌ی پروژه از نتایج این قسـمت اسـتفاده می‌گردد. کد مربوط به این قسـمت به صورت زیر می‌باشد.

```
1.  pMOG2 = cv2.createBackgroundSubtractorMOG2()
2.  all_coordinates = []
3.  all_centroid = []
```

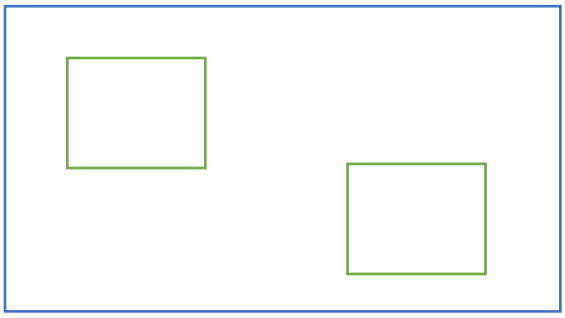
```
bb_color = []
for i in range(len(seq)//1):
    frame2 = seq[i]
    frame = cv2.GaussianBlur(frame2, (9,9), 0)

    fgmaskMOG = pMOG2.apply(frame)
#    fgmaskMOG = cv2.absdiff(frame,background)
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))
    closing = cv2.morphologyEx(fgmaskMOG, cv2.MORPH_CLOSE, kernel)
    img_erosion = cv2.erode(closing, kernel, iterations=3)
    thresh = cv2.threshold(img_erosion, 240, 255, cv2.THRESH_BINARY)[1]


    #convexise
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_TC89_L1)
    contours = cnts[1]


    src = frame2.copy()
    coordinates=[]

    for j in range(len(contours)):
        if cv2.contourArea(contours[j]) < 50:
            continue
        h =cv2.convexHull(contours[j])
        coordinates.append(cv2.boundingRect(h))
    coordinates = np.array(coordinates)
    coordinates = remove_overlapping_bb(coordinates)
    coordinates = remove_corners_bb(coordinates)
    all_coordinates.append(coordinates)

    #calculate centroides
    centroid = centroid_calc(coordinates)

    all_centroid.append(centroid)

    temp = []
    for j in range(len(coordinates)):
        (x, y, w, h)= coordinates[j]
        bb = src[y+h//4:y+3*h//4,x+w//4:x+3*w//4,:];
        color = np.mean(bb,axis=0)
        color = np.floor(np.mean(color, axis=0))
        temp.append(color)

        cv2.rectangle(src, (x, y), (x + w, y + h), (color[0], color[1], color[2])
, 2)
#        cv2.drawContours(src, hull, -1,(0,255,0),3)
    #    fgmaskMOG = cv2.threshold(, 200, 255, cv2.THRESH_BINARY)[1]
    bb_color.append(temp)
    cv2.imshow('frame',src)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
```

3 دنبال کردن اشیای متحرک در ویدیو

در این قسمت قصد داریم اشیای متحرکی که در قسمت قبل با bounding box های متناظر استخراج شد را دنبال کنیم. پیش از این کار ابتدا باید یک سری پیش پردازش بر روی این باکس ها انجام شود.

3_1 پیش پردازش‌ها

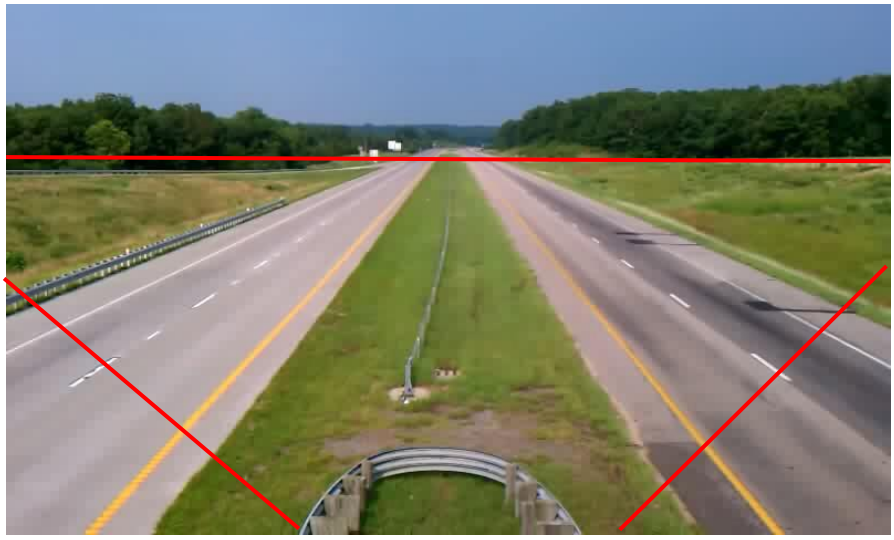
اولین پیش پردازش حذف کردن باکس‌هایی است که درون یک باکس دیگر قرار گرفته‌اند برای این کار از تابع نوشته شده به صورت زیر استفاده گردید به این صورت که اگر مختصات گوشه‌ی چپ بالای یک باکس در باکس دیگر و طول عرض آن نیز کوچک‌تر بود حذف گردد. به عنوان مثال در شکل زیر دو باکس سبز رنگ حذف خواهند شد.



کد این تابع به صورت زیر می‌باشد.

```
def remove_overlapping_bb(coordinates):
    removing_rows = []
    for k in range(len(coordinates)):
        (x1, y1, w1, h1)= coordinates[k]
        for l in range(len(coordinates)):
            if k != l:
                (x2, y2, w2, h2)= coordinates[l]
                if x1< x2  and y1<y2 and (x1+w1)>(x2+w2) and (y1+h1)>(y2+h2):
                    removing_rows.append(l)
    if removing_rows:
        coordinates = np.delete(coordinates, removing_rows, 0)
    return coordinates
```

در مرحله‌ی بعد پیش پردازش لازم بود در برخی از نواحی تصویر مانند جاهایی که خودروها به تصویر ورود یا خروج پیدا می‌کنند و نواحی ای که خودروها بسیار نسبت به دوربین دور هستند باکس‌های متناظر حذف گردند زیرا این باکس‌ها با خطای زیادی همراه هستند. بنابراین مطابق شکل زیر باکس‌های موجود در این نواحی حذف گردید.



شکل 3-1 نواحی حذف باکس ها

تابع مربوط به این قسمت به صورت زیر می‌باشد.

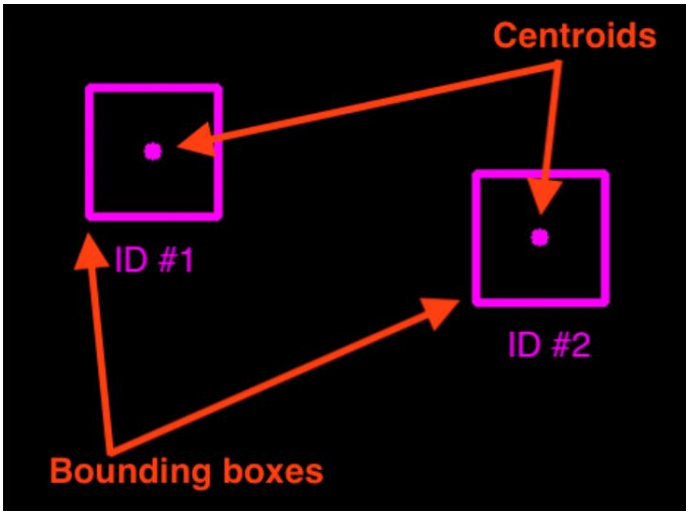
```
def remove_corners_bb(coordinates):
    removing_rows = []
    for k in range(len(coordinates)):
        (x1, y1, w1, h1)= coordinates[k]
        if y1+h1//2<150:
            removing_rows.append(k)
        elif 480- y1 + x1 <300:
            removing_rows.append(k)
        elif (800-x1)/2 + 480- y1 <300:
            removing_rows.append(k)
    coordinates = np.delete(coordinates, removing_rows, 0)
```

در ادامه پس از استخراج باکس‌های مطلوب، تمامی اطلاعات مربوط به مختصات باکس‌ها و مختصات مرکز ثقل آن‌ها در یک ماتریس برای دنبال کردن باکس‌های متناظر در طی ویدیو، ذخیره گردید.

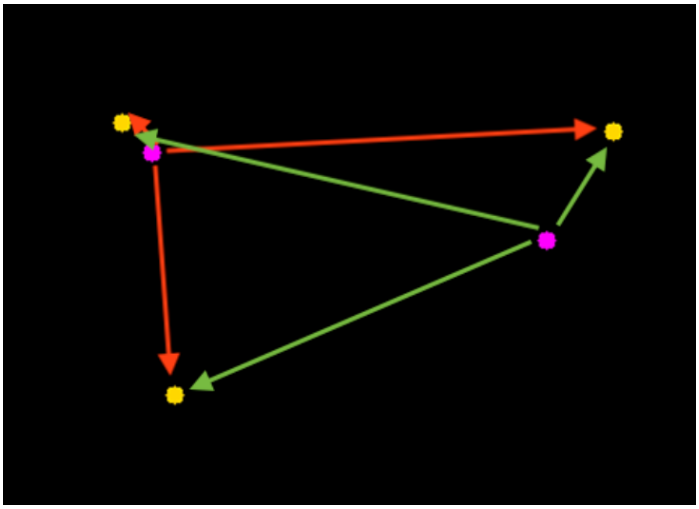
```
    coordinates = np.array(coordinates)
    coordinates = remove_overlapping_bb(coordinates)
    coordinates = remove_corners_bb(coordinates)
    all_coordinates.append(coordinates)

    #calculate centroides
    centroid = centroid_calc(coordinates)

    all_centroid.append(centroid)
```

2_3 دنبال کردن اشیا

برای دنبال کردن اشیا از یک الگوریتم با توجه به حرکت مرکز ثقل باکس‌ها استفاده گردید. به این شکل که برای باکس‌های دو فریم متوالی فاصله‌ی مرکز ثقل‌ها سنجیده می‌شود و در صورتی که فاصله از یک حد کمتر باشد آیدی باکس فریم قبل به باکس فریم جدید نسبت داده می‌شود و اگر هیچ یک از باکس‌های فریم قبل با فریم جدید مطابقت نداشته باشد یک آی دی جدید برای آن در نظر گرفته می‌شود.



شکل 2-3 ایدی های پیدا شده در فریم قبل



شکل 3-3 تناظر ایدی ها

همانطور که در تصویر بالا مشاهده می‌شود دو نقطه‌ی بنفش متعلق به باکس‌ها فریم قبل است. حال در فریم جدید سه مرکز ثقل زرد رنگ برای باکس‌های متناظر موجود است. با بدست آوردن فاصله‌های

مراکز باکس ها مینیمم را برای ایدی متناظر در نظر می‌گیریم. کد مربوط به این قسمت به صورت زیر نوشته شد.

```
import math

frame_vs_ids = []
idx_vs_coordinate = []
correspond_frame = []
idx = 0;
dist_thresh = 30
for i in range(len(seq)-1):
    if idx == 0:
        if all_coordinates[i].any():
            temp_co = []
            for j in range(len(all_coordinates[i])):
                idx += 1
                c = all_centroid[i][j]
                co = all_coordinates[i][j]
                color = bb_color[i][j]
                temp_co.append([idx,c,co,color])
            idx_vs_coordinate.append(temp_co)
            correspond_frame.append(i)
    else:
        last_fr = idx_vs_coordinate[-1]
        temp_co = []
        for j in range(len(all_centroid[i+1])):
            c1=all_centroid[i+1][j]
            closest_coo = all_coordinates[i+1][j]
            color = bb_color[i+1][j]
            mindist = 100

            for idd,c2,_,_ in last_fr:
                dist = math.sqrt((c1[0]-c2[0])**2 + (c1[1]-c2[1])**2)
                if dist <mindist:
                    mindist = dist
                    closest_id = idd
                    closest_cent = c1

            if mindist < dist_thresh:
                temp_co.append([closest_id,closest_cent,closest_coo,color])
            else:
                idx=idx+1
                temp_co.append([idx,c1,closest_coo,color])

        idx_vs_coordinate.append(temp_co)
        correspond_frame.append(i+1)
```

در این تکه کد، فریم‌های متناظر با هر مجموعه باکس و مرکز ثقل و مختصات باکس و رنگ هر باکس ذخیره گردید.

در قسمت بعد با استفاده از ماتریس بدست آمده در مرحله‌ی فوق یک دیکشنری ساخته شد تا متناظر با هر آیدی تمام اطلاعات مربوط به آن آیدی یعنی مختصات باکس‌های موجود در فریم‌های متناظر، مرکز ثقل‌ها و رنگ آن وجود داشته باشد. کد این قسمت به صورت زیر نوشته شد.

```
        id_org_coor_fr = {i:list() for i in range(1,idx+1)}
for i in range(len(correspond_frame)):
    c_f = correspond_frame[i]
    for car_id,org,coor,color in idx_vs_coordinate[i]:
        id_org_coor_fr[car_id].append([org,coor,c_f,color])
```

در قسمت بعد برای حذف آشکارسازی نویز و باکس‌های متوالی بی مورد یک آستانه گذاری انجام می‌شود به این صورت که اگر یک باکس در کمتر از 6 فریم متوالی حضور داشت حذف می‌گردد.

```
delete_idx = []
for i in range(1,len(id_org_coor_fr.keys())+1):
    if len(id_org_coor_fr[i])<6:
        delete_idx.append(i)
        del id_org_coor_fr[i]
```

4 ساخت ویدیوی خلاصه شده

برای ساخت ویدیوی خلاصه شده‌ی نهایی با توجه به دیکشنری ساخته شده در بالا که تمامی تیوب‌های اشـیای متحرک در آن ذخیره گردیده اسـت، اقدام می‌شـود. برای این کار ابتدا بلندترین تیوب زمانی که یک شی در ویدیو حظور داشته پیدا می‌شود. یعنی ویدیوی نهایی ما طولی برابر با بلندترین تیوب خواهد داشت و سایر تیوب‌ها در طول آن قرار داده می‌شود.

برای سـاخت ویدیو با داشـتن مختصـات‌های باکس‌ها یک فیلتر تمام صـفر به جز در نواحی باکس با فریم متناظر باکس ضـرب میشـود و متناظرا یک فیلتر تمام یک به جز در ناحیه‌ی باکس با پس ضـمینه ضـرب می‌شود حال اگر دو تصویر اخیر با یکدیگر جمع شوند تصویر نهایی یکی از فریم‌های ویدیوی خلاصه شده خواهد بود. این کار برای تمامی باکس‌ها انجام می‌شود.

برای نمایش زمان حضـور ماشـین در ویدیوی اصـلی کافیسـت شـماره فریم متناظر با آن ماشـین را که در ویدیوی اصـلی قرار دارد بر frame rate ویدیو که در اینجا 30 اسـت تقسـیم کنیم. کد مربوط به صـورت زیر است.

```
def GetMaxFlow(flows):
    maks=max(flows, key=lambda k: len(flows[k]))
    return maks
frame_shape = seq[0].shape

# font
font = cv2.FONT_HERSHEY_SIMPLEX

# fontScale
```

```
fontScale = 1

# Blue color in BGR
text_color = (255, 255, 255)

# Line thickness of 1 px
thickness = 1
max_fr = GetMaxFlow(id_org_coor_fr)

# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('synopsisVid1.mp4',fourcc, 30, (800,480))

show_original_time = True
show_car_color = False
show_velocity = False

for i in range(max_fr):
    bacg = background.copy()
    for j in id_org_coor_fr.keys():
        mask1 = np.zeros(frame_shape,dtype='uint8')
        mask2 = np.ones(frame_shape,dtype='uint8')
        if i<len(id_org_coor_fr[j]) :
            org, [x,y,w,h], fr_num, color = id_org_coor_fr[j][i]
            frame = seq[fr_num]
            src = frame.copy()
            mask1[y:y+h,x:x+w,:] = 1
            src2 = src*mask1
            mask2[y:y+h,x:x+w,:] = 0
            bacg = bacg*mask2
            bacg = cv2.add(bacg,src2)
            if show_original_time:
                bacg = cv2.putText(bacg, str(int(fr_num/3)/10), (x,y), font, font
Scale, text_color, thickness, cv2.LINE_AA)
            if show_car_color:
                cv2.rectangle(bacg, (x, y), (x + w, y + h), (color[0], color[1],
color[2]), 2)
            if show_velocity:
                bacg = cv2.putText(bacg,str(car_id_velocity[j][0]) , (x,y), font,
 fontScale, text_color, thickness, cv2.LINE_AA)
    cv2.imshow('frame',bacg)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
    out.write(bacg)
out.release()
cv2.destroyAllWindows()
```

یکی از فریم‌های استخراج شده به صورت زیر است که هم ماشین‌ها و هم زمان حضور آن‌ها مشاهده می‌شود.

شکل 1-4 یک فریم از ویدیوی خلاصه شده

این ویدیو با نرخ فریم 20 فریم بر ثانیه 4 ثانیه زمان دارد که نسبت به ویدیوی اصلی که 60 ثانیه با نرخ فریم 30 فریم بر ثانیه می‌باشد بسیار عملکرد خوبی برای خلاصه سازی است.

5 بخش‌های امتیازی

5_1 تشخیص رنگ خودرو

برای تشـــخیص رنگ از یک الگوریتم نســـبتا ســـاده اما کارآمد اســـتفاده گردید. برای این کار در ابتدا که باکس‌های متناظر با هر شی متحرک تشخیص داده می‌شد، یک میانگین بر اساس شدت روشنایی درون باکس گرفته می‌شود و آن رنگ متناظر با خودرو در نظر گرفته می‌شود.

```
bb = src[y+h//4:y+3*h//4,x+w//4:x+3*w//4,:];
color = np.mean(bb,axis=0)
color = np.floor(np.mean(color, axis=0))
temp.append(color)

cv2.rectangle(src, (x, y), (x + w, y + h), (color[0], color[1], color[2]), 2)
```

مثلا در تصویر مشاهده می‌شود که رنگ‌های تخمین زده شده تا حد زیادی متناظر با خودروی مورد نظر است.

شکل 5-1 تعیین رنگ خودروها

در قسمت نهایی کد چند متغییر Boolean تعریف شده است که با صفر و یک کردن هر یک عمل متناظر بر روی ویدیوی خلاصه شده اعمال می‌شود به عنوان مثال با 1 کردن show_car_color باکس متناظر با رنگ خودرو دور خودرو رسم خواهد شد.

```
show_original_time = True
show_car_color = False
show_velocity = False
 if show_original_time:
        bacg = cv2.putText(bacg, str(int(fr_num/3)/10), (x,y), font, fontScale, te
xt_color, thickness, cv2.LINE_AA)
 if show_car_color:
        cv2.rectangle(bacg, (x, y), (x + w, y + h), (color[0], color[1], color[2])
, 2)
 if show_velocity:
        bacg = cv2.putText(bacg,str(car_id_velocity[j][0]) , (x,y), font, fontScal
e, text_color, thickness, cv2.LINE_AA)
```

شکل 5-2 تعیین رنگ خودرو

2_5 شمارش تعداد خودروها و محاسبه‌ی سرعت هر خودرو

برای محاسبه‌ی سرعت خودروها یک ایده‌ی اولیه این است که موقعیت مرکز ثقل در لحظه‌ی ورود و خروج را بدست آورده و بر تعداد فریمی که حضور داشته تقسیم کنیم به این صورت یک تخمینی از فاصله‌ی طی شده تقسیم بر زمان طی شده بدست خواهد آمد. این اعداد را در یک آرایه برای نمایش در ویدیوی اصلی ذخیره می‌کنیم.

با دقت در اعداد بدست آمده در محاسبه‌ی سرعت مشاهده می‌شود که خودروهایی که سرعت مثبت دارند از جنوب به شمال در حال حرکت‌اند و متقابلا خودروهایی که سرعت منفی دارند از شمال به جنوب بنابراین یک تخمین از تعداد خودروهای حرکتی بدست می‌آید.

```
downToUp = 0
UpToDown = 0
car_id_velocity = {i:list() for i in id_org_coor_fr.keys()}
for i in id_org_coor_fr.keys():
    org1,_,c_f1,_ = id_org_coor_fr[i][0]
    org2,_,c_f2,_ = id_org_coor_fr[i][-1]
    velocity = int((org1[1]-org2[1])/(c_f2-c_f1)*50)
    car_id_velocity[i].append(velocity)
    if velocity > 0:
        downToUp +=1
    else:
        UpToDown +=1
print('Number of Cars go Up: ', downToUp)
print('Number of Cars go Down: ', UpToDown)
```

نتیجه‌ی شمارش به صورت زیر بدست آمد:

```
Number of Cars go Up:  21
Number of Cars go Down:  24
```

برای نمایش سرعت نیز در کد اصلی به صورت زیر با قرار دادن متغییر Boolean متناظر برابر مقدار 1، سرعت‌ها نمایش داده شد.

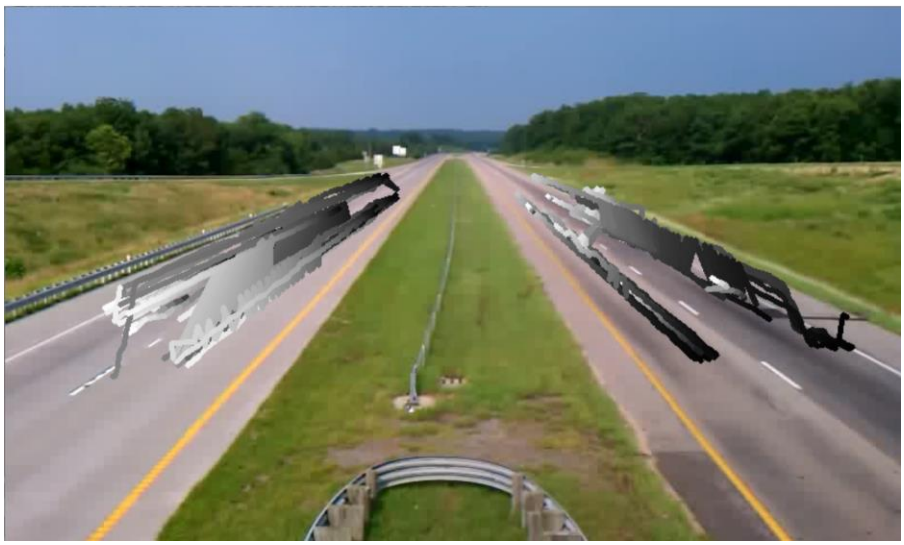
شکل 3-5 نمایش سرعت

در تصویر هم مشاهده می‌شود که خودروهای دارای سرعت منفی از شمال به جنوب در حال حرکتند.

3_5 تشخیص مسیر حرکت و ترسیم آن

برای تشخیص مسیر حرکت و رسم آن با توجه به داشتن اطلاعات مختصات‌های باکس‌های متناظر با هر خودرو در هر فریم این کار به راحتی امکان پذیر است. با استفاده از کد زیر تمامی گرادیان لاین ها برای خودروهای عبوری رسم شده است.

```
gr_line = cv2.imread('background1.jpg')

for i in id_org_coor_fr.keys():
    start = True
    steps = len(id_org_coor_fr[i])
    brightness = 255//steps
    c = 0
    for org,_,_,_ in id_org_coor_fr[i]:
        c += brightness
        if start:
            org1 = org
            start = False
            continue
        org2 = org
        cv2.line(gr_line,(org1[0],org1[1]),(org2[0],org2[1]),(c,c,c),thickness = 3)
        org1 = org2

cv2.imshow('Gradient Lines',gr_line)
k = cv2.waitKey(0)
cv2.destroyAllWindows()
```



شکل 4-5 رسم مسیر حرکت خودروها

6 مراجع

.1 Pritch, Y., A. Rav-Acha, and S. Peleg, *Nonchronological video synopsis and indexing.* IEEE transactions on pattern analysis and machine intelligence, 2008. **30**(11): p. 1971-1984 %@ 0162-8828.

.2 Rav-Acha, A., Y. Pritch, and S. Peleg. *Making a long video short: Dynamic video synopsis*. 2006. IEEE.

.3 Xu, M., et al. *A set theoretical method for video synopsis*. 2008.